Reinforcement Learning

An Intuitive Introduction with Mathematics and Algorithms

Ahmed BADI

Mathematics & Machine Learning Enthusiast

ahmedbadi905@gmail.com
linkedin.com/in/badi-ahmed

January 2026

Overview

Introduction

What is Reinforcement Learning?

- RL is about **learning from interaction** with an environment to make better decisions over time.

What is Reinforcement Learning?

- RL is about **learning from interaction** with an environment to make better decisions over time.
- Agent repeatedly:
  - observes a state,
  - takes an action,
  - receives a numerical reward,
  - moves to a new state.

What is Reinforcement Learning?

- RL is about **learning from interaction** with an environment to make better decisions over time.
- Agent repeatedly:
  - observes a state,
  - takes an action,
  - receives a numerical reward,
  - moves to a new state.
- **No labeled input-output pairs**: learning is driven by rewards, often delayed in time.

What is Reinforcement Learning?

- RL is about **learning from interaction** with an environment to make better decisions over time.
- Agent repeatedly:
  - observes a state,
  - takes an action,
  - receives a numerical reward,
  - moves to a new state.
- **No labeled input-output pairs**: learning is driven by rewards, often delayed in time.
- Goal: learn a behaviour that **maximizes cumulative reward**.

RL vs Supervised Learning

Supervised Learning

- Data: input-output pairs (x, y) with correct labels.
- Objective: minimize prediction error on y.

RL vs Supervised Learning

Supervised Learning

- Data: input-output pairs (x, y) with correct labels.
- Objective: minimize prediction error on y.

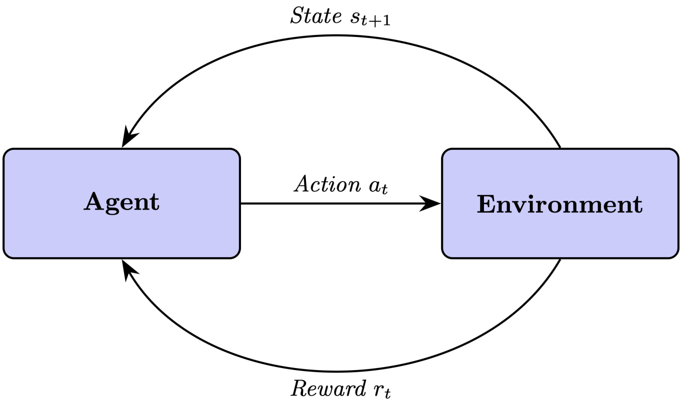
Reinforcement Learning

- Data: sequences of states, actions, rewards.
- No explicit "correct action" given.
- Objective: maximize expected return (long-term reward), not immediate accuracy.

Motivating Applications

- **Game playing**: chess, Go, Atari, complex strategy games.
- **Recommendation systems**: optimize long-term engagement (videos, products).
- **Robotics and control**: autonomous driving, robotic arms, drones.
- **Operations research**: inventory management, job scheduling, resource allocation.

The RL Loop (Figure)



Agent observes the state, selects an action, receives a reward and a new state from the environment.

RL Framework: MDPs

Markov Decision Process (MDP)

RL problems are often modeled as an MDP $(\mathcal{S}, \mathcal{A}, P, R, \gamma)$:

- $\mathcal{S}$: set of states.
- $\mathcal{A}$: set of actions.
- $P(s' \mid s, a)$: transition probability to s' from s under action a.
- $R(s, a, s')$: reward for moving from s to s' via a.
- $\gamma \in [0, 1)$: discount factor for future rewards.

Markov property: next state depends only on current state and action, not full history.

Interaction in an MDP

At each time step t:

1. Agent observes state $S_t \in \mathcal{S}$.
2. Chooses action $A_t \in \mathcal{A}$ according to a policy.
3. Environment returns reward R_{t+1} and next state S_{t+1}.

Policy

A policy $\pi(a \mid s)$ is the agent's strategy:

$$\pi(a \mid s) = P(A_t = a \mid S_t = s)$$

It can be deterministic or stochastic.

Return and Discounting

Return

The cumulative discounted reward from time t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

Return and Discounting

Return

The cumulative discounted reward from time t:

$$G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+1+k}$$

- γ close to 1: future rewards matter a lot (long-term view).
- Smaller γ: focus more on immediate rewards.
- Discounting keeps G_t finite and expresses uncertainty about the future.

Value Functions & Bellman Equation

State and Action Value Functions

State-Value Function $v_\pi(s)$

Expected return starting in state s and following policy π:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Measures how good a state is under π.

State and Action Value Functions

State-Value Function $v_\pi(s)$

Expected return starting in state s and following policy π:

$$v_\pi(s) = \mathbb{E}_\pi \left[G_t \mid S_t = s \right]$$

Measures how good a state is under π.

Action-Value Function $q_\pi(s, a)$

Expected return starting in state s, taking action a, then following π:

$$q_\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a \right]$$

Measures how good a state-action pair is under π.

Bellman Expectation Equations

For $v_\pi(s)$

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma\, v_\pi(s')\right]$$

Bellman Expectation Equations

For $v_\pi(s)$

$$v_\pi(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma\, v_\pi(s')\right]$$

For $q_\pi(s, a)$

$$q_\pi(s, a) = \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s', a') \right]$$

These express value as immediate reward plus discounted value of successor states.

Optimal Value Functions

- An optimal policy $\pi^\star$ maximizes expected return.
- Optimal value functions:

$$v_\star(s) = \max_\pi v_\pi(s), \quad q_\star(s,a) = \max_\pi q_\pi(s,a).$$

- They satisfy Bellman optimality equations:

$$v_\star(s) = \max_a \sum_{s',r} P(s', r \mid s, a)[r + \gamma v_\star(s')]$$

$$q_\star(s,a) = \sum_{s',r} P(s', r \mid s, a)[r + \gamma \max_{a'} q_\star(s', a')]$$

Dynamic Programming Methods

Policy Evaluation

Given a fixed policy π, we can compute v_π by iteration:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_k(s') \right]$$

Policy Evaluation

Given a fixed policy π, we can compute v_π by iteration:

$$v_{k+1}(s) = \sum_a \pi(a \mid s) \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_k(s') \right]$$

- Start from an initial guess $v_0(s)$.
- Iterate until v_k converges.
- Requires full knowledge of P and R (model-based).

Policy Iteration

1. Initialize policy π_0 arbitrarily.
2. **Policy evaluation:** compute v_{π_k}.
3. **Policy improvement:** update policy greedily:

$$\pi_{k+1}(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_{\pi_k}(s')\right]$$

4. Repeat until policy stabilizes (converges to $\pi^\star$).

Value Iteration

Combined Evaluation & Improvement

$$v_{k+1}(s) = \max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_k(s') \right]$$

Value Iteration

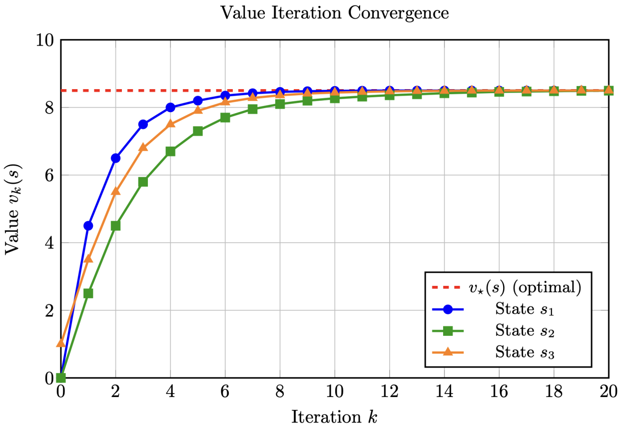
Combined Evaluation & Improvement

$$v_{k+1}(s) = \max_a \sum_{s',r} P(s', r \mid s, a) \left[r + \gamma v_k(s') \right]$$

- Directly pushes v_k towards $v_\star$.
- After convergence, derive optimal policy:

$$\pi^\star(s) = \arg\max_a \sum_{s',r} P(s', r \mid s, a)[r + \gamma v_\star(s')]$$

Value Iteration Convergence (Figure)



Example of value iteration converging to the optimal value function over iterations.

Monte Carlo and TD Learning

Monte Carlo Value Estimation

State-Value Estimation

For policy π:

$$v_\pi(s) \approx \frac{1}{N(s)} \sum_{i=1}^{N(s)} G^{(i)}(s)$$

where $G^{(i)}(s)$ is the return after the i-th visit to s.

Monte Carlo Value Estimation

State-Value Estimation

For policy π:

$$v_\pi(s) \approx \frac{1}{N(s)} \sum_{i=1}^{N(s)} G^{(i)}(s)$$

where $G^{(i)}(s)$ is the return after the i-th visit to s.

- Requires complete episodes.
- Does not need P or R (model-free).
- Simple and unbiased, but cannot update before an episode ends.

Temporal-Difference Learning: TD(0)

TD(0) Update

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

δ_t is the TD error.

Temporal-Difference Learning: TD(0)

TD(0) Update

$$V(S_t) \leftarrow V(S_t) + \alpha \left[R_{t+1} + \gamma V(S_{t+1}) - V(S_t) \right]$$

$$\delta_t = R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$$

δ_t is the TD error.

- Learn **online**, step by step.
- Use **bootstrapping**: update from estimates.
- Often more data-efficient than plain Monte Carlo.

Q-Learning and Exploration

Q-Learning: Model-Free Control

Q-Learning Update

Maintain $Q(s, a)$ and update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right]$$

Q-Learning: Model-Free Control

Q-Learning Update

Maintain $Q(s, a)$ and update:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_{a'} Q(S_{t+1}, a') - Q(S_t, A_t) \right]$$

- Learns $Q_\star(s, a)$ directly from experience.
- Does not require knowledge of transition probabilities or rewards.
- Greedy policy: $\pi(s) = \arg\max_a Q(s, a)$.

Exploration vs Exploitation

RL must balance:

- **Exploration**: try new actions to discover better strategies.
- **Exploitation**: choose actions known to yield high reward.

Exploration vs Exploitation

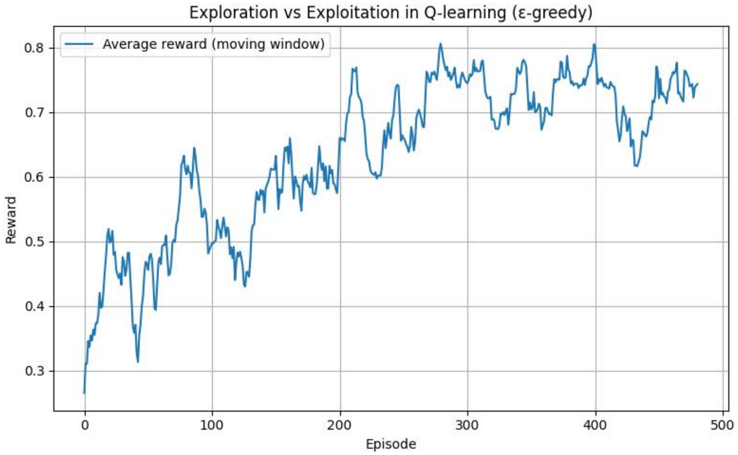
RL must balance:

- **Exploration**: try new actions to discover better strategies.
- **Exploitation**: choose actions known to yield high reward.

ϵ-Greedy Policy

- With probability ϵ: choose a random action (explore).
- With probability $1 - \epsilon$: choose $\arg\max_a Q(s, a)$ (exploit).

Q-Learning Learning Curve (Figure)



Example: cumulative reward improving over episodes as Q-learning converges.

Algorithms Overview

Core Tabular RL Algorithms

Method	What it estimates	Needs model?	Typical use
Policy Evaluation	$v_\pi(s)$ for fixed π	Yes (full MDP)	Analyze given policy
Policy Iteration	Optimal $\pi^\star$ via v_π	Yes	Exact solution for small MDPs
Value Iteration	$v_\star(s)$ and $\pi^\star$	Yes	Exact planning
Monte Carlo	$v_\pi(s)$ or $q_\pi(s,a)$ from episodes	No	Episodic tasks, unknown dynamics
TD(0)	$v_\pi(s)$ online via TD error	No	Online prediction
Q-Learning	$Q_\star(s,a)$ via off-policy TD control	No	Model-free control

Beyond Tabular RL

Policy Gradient (High-Level)

- Instead of learning value functions, directly parametrize the policy $\pi_\theta(a \mid s)$.
- Objective: maximize expected return

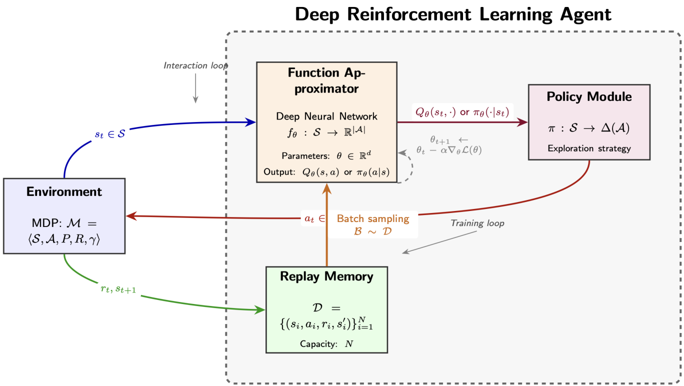
$$J(\theta) = \mathbb{E}_{\pi_\theta}[G_0].$$

- Policy gradient methods estimate $\nabla_\theta J(\theta)$ and update

$$\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta).$$

- REINFORCE uses Monte Carlo returns to estimate this gradient.

Deep Reinforcement Learning (Figure)



Training objective: Minimize loss $\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}[(y - Q_\theta(s,a))^2]$ where $y = r + \gamma \max_{a'} Q_{\theta^-}(s',a')$ (DQN).
Key mechanisms: Experience replay for sample efficiency, target network θ^- for stability, ϵ-greedy for exploration-exploitation trade-off.

Deep RL uses neural networks to approximate value functions or policies in large/continuous state spaces.

Practical Considerations and Conclusion

Practical Challenges

- **Sample efficiency**: many interactions can be required to learn a good policy.
- **Exploration**: safe and effective exploration is non-trivial.
- **Credit assignment**: which past actions caused current reward?
- **Function approximation**: generalization in large or continuous state spaces.

Good Practice Tips

- Start with small, tabular problems (gridworlds, bandits) to build intuition.
- Tune learning rate α, discount factor γ, and exploration schedule carefully.
- Monitor learning curves (e.g., average reward per episode).
- For policy gradients, use baselines and variance reduction techniques.

Conclusion

- RL offers a powerful framework for learning to act under uncertainty through trial and error.
- MDPs and Bellman equations provide the mathematical foundation.
- Dynamic programming, Monte Carlo, TD learning, and Q-learning form the core tabular toolbox.
- Policy gradients and deep RL extend these ideas to complex, high-dimensional problems.

Thank you!

Questions?

ahmedbadi905@gmail.com
linkedin.com/in/badi-ahmed